LAB 10

Advanced Procedures

Syed Muhammad Faheem	20K1054	3E
STUDENT NAME	ROLL NO	SEC

LAB ENGINEER'S SIGNATURE & DATE

MARKS AWARDED: /

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Prepared by: Qurat ul ain

Lab Session 10: Advanced Procedures

Learning Objectives

- Implementing procedures using stack frame
- Using stack parameters in procedures
- Passing value type and reference type parameters

Stack Applications

There are several important uses of runtime stacks in programs:

1. A stack makes a convenient temporary save area for registers when they are used for more than one purpose. After they are modified, they *can* be restored to their original values.
2. When the CALL instruction executes, the CPU saves the current subroutine's return address on the stack.
3. When calling a subroutine, you pass input values called arguments by pushing them on the stack.
4. The stack provides temporary storage for local variables inside subroutines.

Stack Parameters

Passing by value

When an argument is passed by value, a copy of the value is pushed on the stack.

EXAMPLE # 01:

```
  .data
var1      DWORD       5
var2      DWORD       6
  .code push
  var2   push
  var1  call
  AddTwo
  exit
  AddTwo  PROC  push
  ebp   mov   ebp, esp
```

```
mov  eax, [ebp + 12]
add   eax,  [ebp  +  8]
pop   ebp
ret
AddTwo ENDP
```

Explicit stack parameters

When stack parameters are referenced with expressions such as [ebp+8], we call them explicit stack parameters.

Example 2:

```
.data
var1        DWORD        5
var2        DWORD        6
y_param          EQU     [ebp + 12]
x_param          EQU     [ebp+ 8]
.code  push
var2   push
var1   call
AddTwo
exit
AddTwo PROC
push ebp   mov
ebp, esp  mov
eax, y_param
add eax, x_param
pop ebp
ret
AddTwo ENDP
```

- **Passing by reference**

  An argument passed by reference consists of the offset of an object to be passed.

EXAMPLE # 03:

```
.data
count = 10
arr          WORD count DUP (?)
.code push
OFFSET arr  push
count
call ArrayFill exit
ArrayFill PROC
push ebp mov
ebp, esp  pushad
mov esi, [ebp + 12]
mov ecx, [ebp + 8]
cmp ecx, 0 je L2 L1:
mov    eax, 100h call
RandomRange  mov
[esi], ax add esi,
TYPE WORD   loop
L1 L2:  popad pop
ebp
ret 8
ArrayFill ENDP
```

LEA Instruction

LEA instruction returns the effective address of an indirect operand. Offsets of indirect operands are calculated at runtime.

EXAMPLE # 04:

```
.code call
        makeArray
exit
makeArray  PROC
push  ebp  mov
```

```
 ebp, esp  sub  esp,
32  lea   esi,  [ebp  -
30]  mov ecx,30
L1:
mov    BYTE PTR [esi], '*'
inc    esi
loop   L1
add    esp, 32
pop    ebp ret
makeArray       ENDP
```

ENTER & LEAVE Instructions

Enter instruction automatically creates stack frame for a called Procedure. Leave instruction reverses the effect of enter instruction.

EXAMPLE # 05:

```
.data
var1  DWORD          5
var2  DWORD          6
.code  push
var2  push var1
call AddTwo  exit
AddTwo PROC  enter
0, 0 mov
       eax, [ebp + 12]
add  eax, [ebp + 8]
leave ret
AddTwo ENDP
```

Local Variables

In MASM Assembly Language, local variables are created at runtime stack, below the base pointer (EBP).

EXAMPLE # 06:

```
.code
call     MySub   exit
MySub PROC
```

```
push    ebp
mov     ebp, esp
sub     esp, 8
mov     DWORD       PTR [ebp - 4], 10       ; first parameter
mov     DWORD       PTR [ebp - 8], 20       ; second parameter
mov     esp, ebp
pop     ebp
ret
MySub ENDP
```

LOCAL Directive

LOCAL directive declares one or more local variables by name, assigning them size attributes.

EXAMPLE # 07:

```
.code call
LocalProc  exit
LocalProc PROC LOCAL
temp : DWORD mov
       temp, 5
mov    eax, temp   ret
LocalProc ENDP
```

Recursive Procedures

Recursive procedures are those that call themselves to perform some task.

EXAMPLE # 08:

```
.code L1:  mov
ecx, 5
mov eax, 0 call CalcSum
       call
WriteDec
call  crlf  exit
CalcSum        PROC   cmp
       ecx, 0
```

```
jz          L2
add  eax, ecx  dec

ecx  call  CalcSum

  L2: ret
CalcSum          ENDP
```

INVOKE Directive

The INVOKE directive pushes arguments on the stack and calls a procedure. INVOKE is a convenient replacement for the CALL instruction because it lets you pass multiple arguments using a single line of code. Here is the general syntax:

INVOKE procedureName [, argumentList]

For example: push

TYPE array push

LENGTHOF array

push OFFSET array

call DumpArray is

equal to

INVOKE DumpArray, OFFSET array, LENGTHOF array, TYPE array

ADDR Operator

The ADDR operator can be used to pass a pointer argument when calling a procedure using INVOKE. The following INVOKE statement, for example, passes the address of myArrayto the FillArrayprocedure: INVOKE FillArray, ADDR myArray

PROC Directive

Syntax of the PROC Directive

The PROC directive has the following basic syntax:

Label PROC [attributes] [USES reglist], parameter_list

The PROC directive permits you to declare a procedure with a comma-separated list of named parameters.

Example: The FillArray procedure receives a pointer to an array of bytes:

FillArray PROC,

```
pArray:PTR BYTE
. . .
FillArray ENDP
```

PROTO Directive

The PROTO directive creates a prototype for an existing procedure. A prototype declares a procedure's name and parameter list. It allows you to call a procedure before defining it and to verify that the number and types of arguments match the procedure definition.

```
MySub PROTO ; procedure prototype

        .
        INVOKE MySub ; procedure call
        .
        MySub PROC          ; procedure implementation
        .
        MySub ENDP
```

Exercises:

1. Write a program which contains a procedure named **BubbleSort** that sorts an array which is passed through a stack using indirect addressing.

2. Write a program which contains a procedure named **TakeInput** which takes input numbers from user and call a procedure named **Armstrong** which checks either a number is an Armstrong number or not and display the answer on console by calling another function **Display**. (Also show ESP values during nested function calls)

3. Write a program which contains a procedure named **Reverse** that reverse the string using recursion.

4. Write a program which contains a procedure named **LocalSquare** . The procedure must declare a local variable. Initialize this variable by taking an input value from the user and then display its square. Use **ENTER *& LEAVE*** instructions to allocate and de-allocate the local variable.

5. Write a program that calculates factorial of a given number ***n***. Make a recursive procedure named **Fact** that takes n as an input parameter.

6. Write a program to take 4 input numbers from the users. Then make two procedures **CheckPrime** and **LargestPrime**. The program should first check if a given number is a prime number or not. If all of the input numbers are prime numbers then the program should call the procedure LargestPrime.

CheckPrime: This procedure tests if a number is prime or not

LargestPrime: This procedure finds and displays the largest of the four prime numbers.

Task 1:

```
include irvine32.inc
include macros.inc
.stack 100h
.model small
.data
arr1 dword 5,4,3,2,1
.code
main proc
mov ecx, lengthof arr1
push ecx
mov esi,offset arr1
push esi
call BubbleSort
exit
main endp
BubbleSort proc
push ebp
mov ebp,esp
mov ecx,[ebp+12]
L1:
mov esi,[ebp+8]
mov edi,[ebp+8]
add edi,4
mov ebx,ecx
dec ebx
mov ecx,ebx
inc ebx
L2:
cmp ecx,0
JZ L4
mov eax,[esi]
cmp eax,[edi]
JNS L3
add edi,4
add esi,4
loop L2
L3:
mov edx,[edi]
mov [esi],edx
mov [edi],eax
dec ecx
JECXZ L4
add edi,4
add esi,4
jmp L2
L4:
mov ecx,ebx
loop L1
mov ecx,[ebp+12]
mov esi,[ebp+8]
mWrite "The sorted array is: "
L5:
mov eax,[esi]
call WriteDec
mWrite " "
add esi,4
loop L5
pop ebp
ret
BubbleSort endp
end main
```

```
Microsoft Visual Studio Debug Console
The sorted array is: 1 2 3 4 5
C:\Users\Faheem\source\repos\Prac\Debug\Prac.exe (process 28604) exited with code 0.
Press any key to close this window . . .
```

Task 2:

```
include irvine32.inc
include macros.inc
.stack 100h
.model small
.data
var dword ?
divisor dword 10
sum dword ?
.code
main proc
call TakeInput
exit
main endp
Display Proc
push ebp
mov ebp,esp
mov eax,[ebp+8]
call WriteDec
mWrite " is an Armstrong Number!"
pop ebp
ret 4
Display endp
TakeInput proc
mWrite "Enter the number to check: "
call ReadInt
push divisor
push eax
call CheckArmstrong
ret
TakeInput endp
CheckArmstrong proc
push ebp
mov ebp,esp
sub esp,4
mov eax,[ebp+8]
mov dword ptr[ebp-4],0
L1:
mov ecx,[ebp+12]
mov edx,0
div ecx
mov ebx,eax
mov eax,edx
mov ecx,edx
mul ecx
mul ecx
add [ebp-4],eax
mov eax,ebx
cmp eax,0
JNZ L1
mov eax,[ebp-4]
mov ebx,[ebp+8]
cmp eax,ebx
JZ L3
mWrite "The Number entered is not an Armstrong Number!"
pop ebp
add esp,4
ret 8
L3:
push eax
call Display
add esp,4
pop ebp
ret 8
```

```
Microsoft Visual Studio Debug Console
Enter the number to check: 153
153 is an Armstrong Number!
C:\Users\Faheem\source\repos\Prac\Debug\Prac.exe (process 8648) exited wit
Press any key to close this window . . .
```

Task 3:

```
include irvine32.inc
include macros.inc
.model small
.stack 100h
.data
Str1 byte 25 dup(?)
.code
main proc
mov ecx,lengthof Str1
mWrite "Enter the string to reverse: "
mov edx,offset Str1
call ReadString
mov [Str1+eax],0
mov ebx,eax
dec ebx
shr eax,1
mov ecx,eax
mov eax,ebx
mov ebx,0
call Reverse
exit
main endp

Reverse proc
cmp ecx,0
JNE L1
mWrite "The Reversed String is: "
mov edx,offset Str1
call WriteString
ret
L1:
mov esi,offset Str1
add esi,eax
mov edi,offset Str1
add edi,ebx
mov edx,0
mov dl,[esi]
xchg dl,[edi]
mov [esi],dl
inc ebx
dec eax
dec ecx
call Reverse
ret
Reverse endp
end main
```

```
Microsoft Visual Studio Debug Console
Enter the string to reverse: faheem
The Reversed String is: meehaf
C:\Users\Faheem\source\repos\Prac\Debug\Prac.exe (process 4976) exited with code
Press any key to close this window . . .
```

Task 4:

```
include irvine32.inc
include macros.inc
.model small
.stack 100h
.data
.code
main proc
call LocalSquare
exit
main endp
LocalSquare proc
enter 1,0
mWrite "Enter the number to get the square of: "
call ReadInt
mul eax
mov [ebp-4],eax
mWrite "The Square of the entered number is: "
call WriteDec
leave
ret
LocalSquare endp
end main
```

```
Microsoft Visual Studio Debug Console
Enter the number to get the square of: 8
The Square of the entered number is: 64
C:\Users\Faheem\source\repos\Prac\Debug\Prac.exe (process 17964) exited with code 0.
Press any key to close this window . . .
```

Task 5:

```
include irvine32.inc
include macros.inc
.stack 100h
.model small
.data
num dword ?
.code
main proc
mWrite "Enter the number you want to take the factorial of: "
call ReadInt
mov ecx,eax

mov eax,1

call Factorial
exit
main endp

Factorial proc
cmp ecx,0
JNE L1
mWrite "The factorial of the given number is: "
call WriteDec
ret
L1:
mul ecx
dec ecx
call Factorial
ret
Factorial endp
end main
```

```
Microsoft Visual Studio Debug Console
Enter the number you want to take the factorial of: 6
The factorial of the given number is: 720
C:\Users\Faheem\source\repos\Prac\Debug\Prac.exe (process 7
Press any key to close this window . . .
```

Task 6:

```
include irvine32.inc
include macros.inc
.stack 100h
.model small
.data
arr dword 4 dup(?)
check dword 1
indexes dword 4 dup(?)
.code
main proc
mWrite "Enter the 4 values to check the prime number status: "
mov esi, offset arr
mov ecx,4
L1:
call ReadInt
mov [esi],eax
add esi,4
loop L1
call CheckPrime
exit
main endp

LargestPrime proc
cmp check,0
JNZ L7
call Crlf
mWrite "All the values of the array were not prime!"
L7:
call Crlf
mWrite "The Prime Values are: "
mov ecx,Lengthof indexes
mov esi,offset indexes
mov ebx,0
L6:
cmp ebx,[esi]
JB L10
L11:
mov eax,[esi]
cmp eax,0
JZ L9
call WriteDec
mWrite " "
L9:
add esi,4
loop L6
call Crlf
mWrite "The Largest Prime Value is: "
mov eax,ebx
call WriteDec
ret
L10:
mov ebx,[esi]
jmp L11
ret
LargestPrime endp

CheckPrime proc
mov ecx,4
mov edi,offset indexes
mov esi,offset arr
L2:
mov ebx,ecx
mov ecx,0
```

```
Microsoft Visual Studio Debug Console
Enter the 4 values to check the prime number status: 2
3
4
7

All the values of the array were not prime!
The Prime Values are: 2 3 7
The Largest Prime Value is: 7
C:\Users\Faheem\source\repos\Prac\Debug\Prac.exe (process 21548) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Auto
le when debugging stops.
Press any key to close this window . . .
```